CrystalBall: Statically Analyzing Runtime Behavior via Deep Sequence Learning

Stephen Zekany
Daniel Rings
Nathan Harada
Michael A. Laurenzano
Lingjia Tang
Jason Mars

Introduction

- Why analyze runtime behavior?
- How to analyze it for software lifecycle? – Hot Paths (1 in a million)
- Path profiling:
  - Dynamic Profiling:

    Digital Mars C++
  - Group functions that call each other
  - Static Profiling:

    Predict runtime behavior before the program runs
  - Applications - Branch Prediction, Trace formation, Basic Block placement optimization

Why not Dynamic Profiling?

- Needs representative production environment
- Computationally Expensive
- In for a penny, in for a pound

Static Profiling – CrystalBall

- Program behavior is latent within instructions
- Higher the quality of static analysis => better runtime prediction
- Can leverage large amount of data
- Language independent – uses Intermediate Representation (IR)
- IR – Semantic + Low - level Ops

  Compilers - GCC, LLVM (Low Level Virtual Machine)
- Sequence of blocks => use RNN

Intermediate Representation

C++ Function -

```
int mul_add(int x, int y, int z){
    return x * y + z;
}
```

IR -

```
define i32 @mul_add(i32 %x, i32 %y, i32 %z) {
    entry:
        %tmp = mul i32 %x, %y
        %tmp2 = add i32 %tmp, %z
        ret i32 %tmp2
}
```

Basic Block

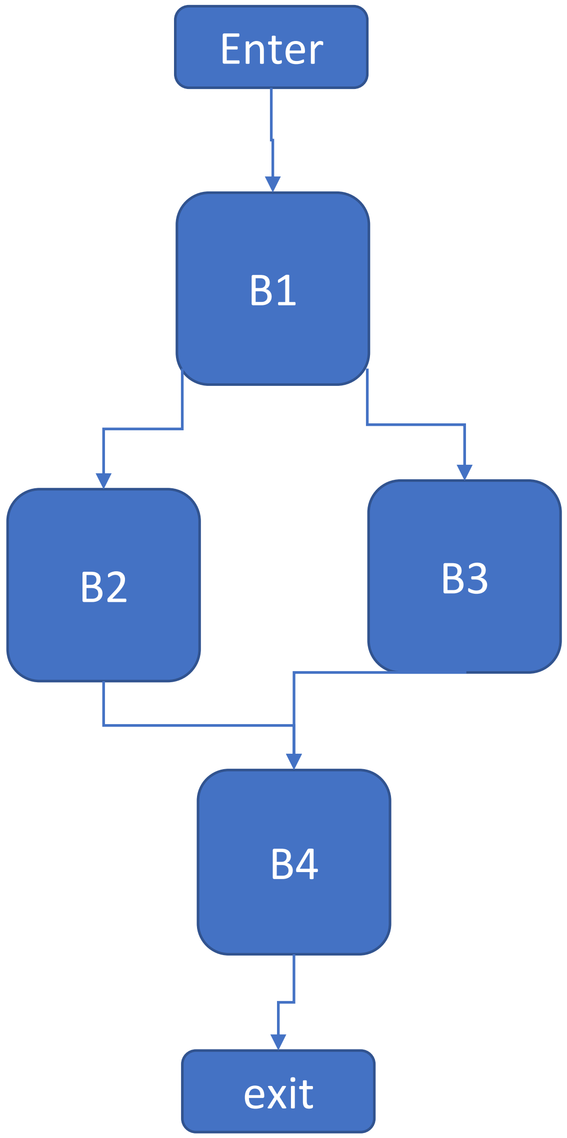
Source Code:

```
w = 0;
x = x + y;
y = 0;
if ( x > z) {
        y= x;
        x++;
}
else{
        y = z;
        z++;
}
w = x + z;
```

Basic Blocks:

w = 0;
x = x + y;
y = 0;
if (x > z)
	B1

y= x; x++;	**B2**
y = z; z++;	**B3**
w = x + z;	**B4**



Ball Larus Path Profiling

- ➢ Convert each function to Directed Acyclic Graph (DAG)
- ➢ Back edges are removed in DFS
- ➢ Unique sum of edge weight for a pat

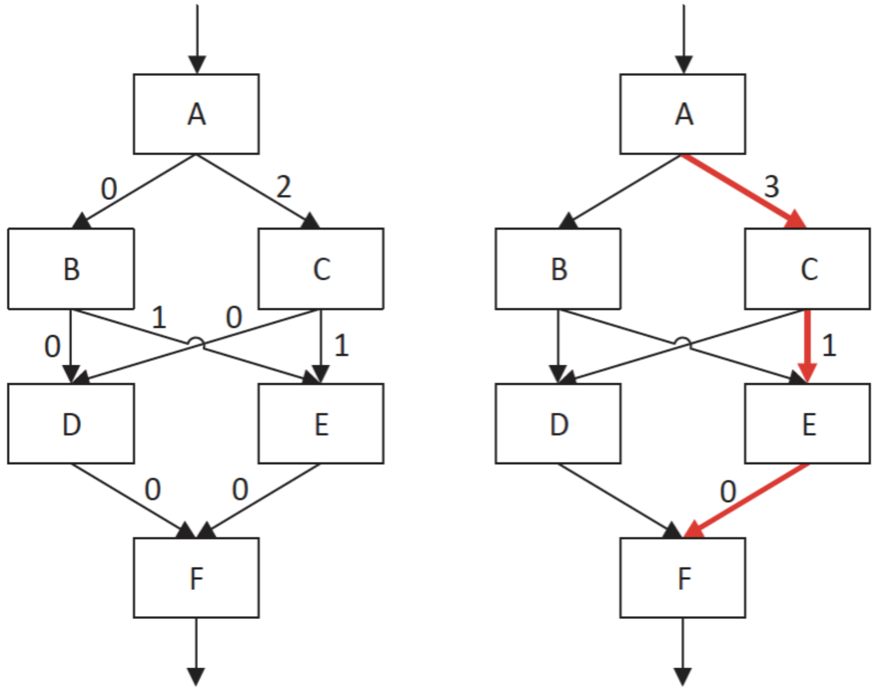


Fig. 1: Example of function path enumeration using Ball-Larus algorithm (left - edge weights between basic blocks, right - example of path reconstruction)

Performance Metrics

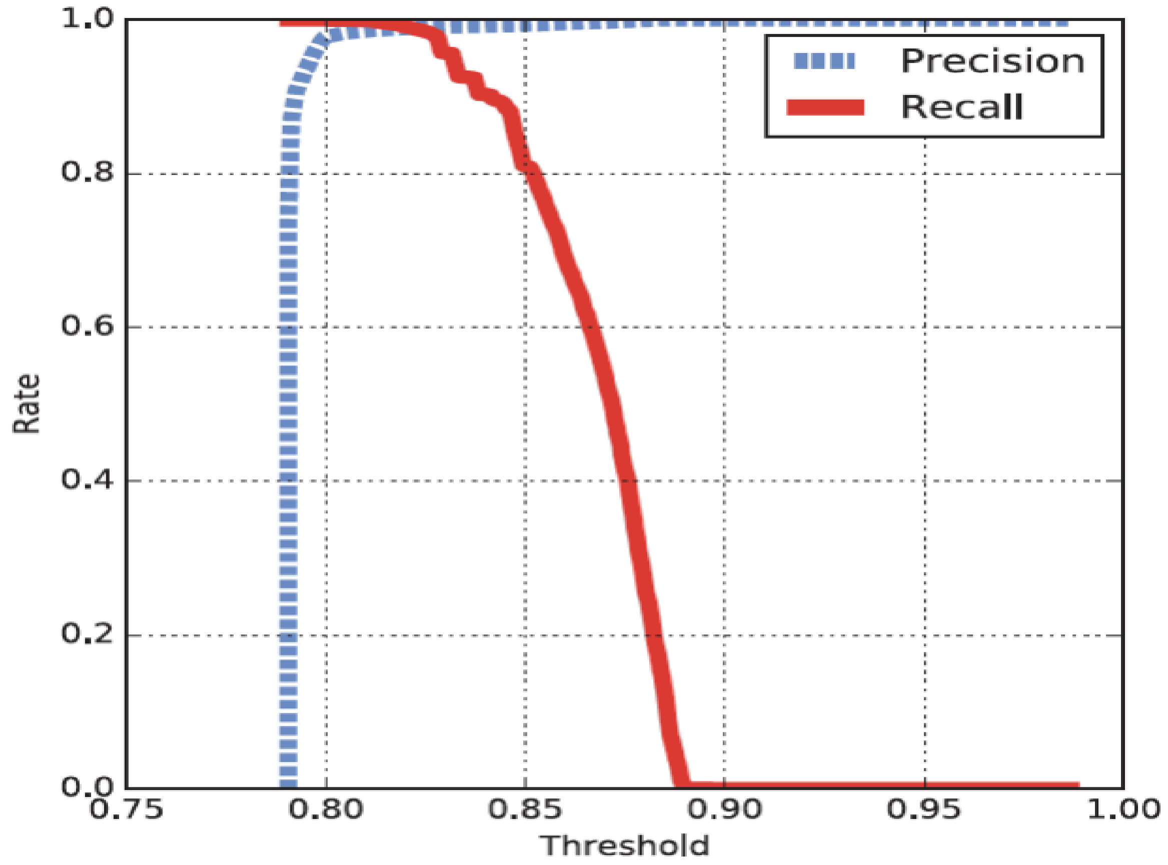
Confusion Matrix:

Predicted

Actual

	+ve	-ve
+ve	TP	FN
-ve	FP	TN

- Precision = TP/ (TP + FP)
- Recall = TP/(TP+FN)
- F1 – measure = 2 * Precision * Recall /(Precision + Recall)



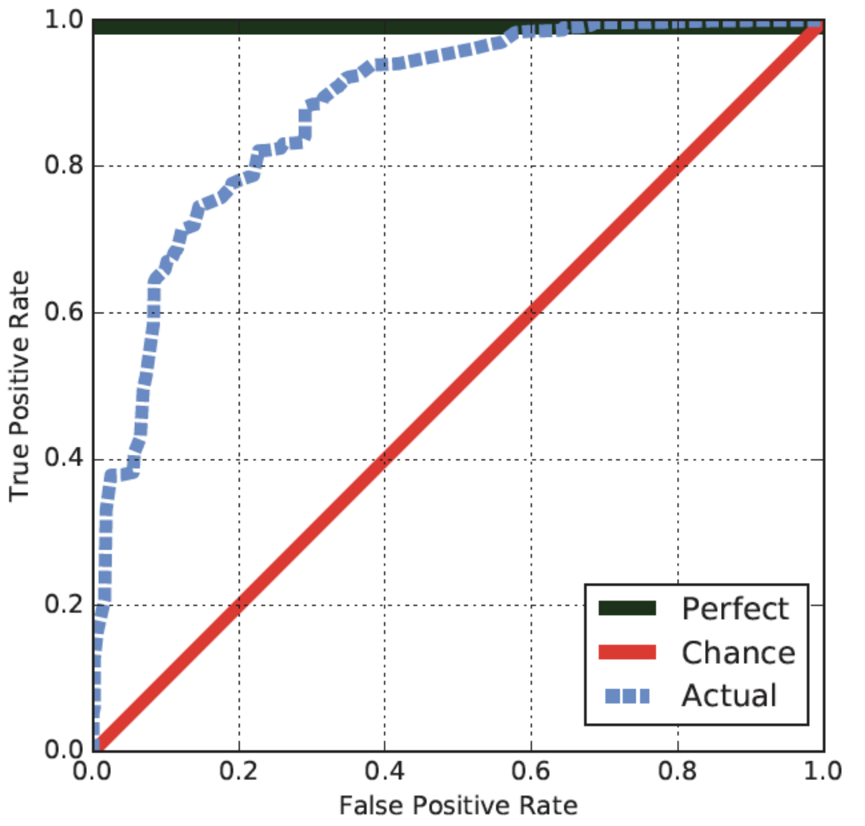
Solution – AUROC (Area Under ROC)

TPR (Recall) = TP/ (TP + FN)

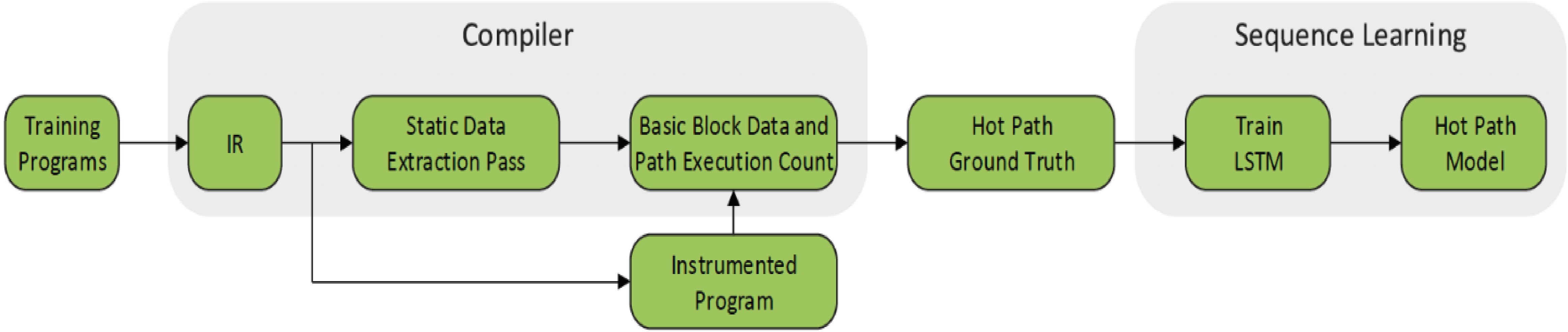
FPR = FP/(FP+TN)

TPR = FPR (Random)

More area => better classifier



Crystal Ball - Overview



Crystal Ball - Implementation

- Data Collection: Using Profiling Instrumentation

- Static Data Extraction
  - Basic Block to feature vector

- Path Sampling –
  - Include all Hot Paths
  - Proportional Sampling for Cold paths
  - Equal number of Cold paths for every function (2000)

- Training: leave-one-program-out

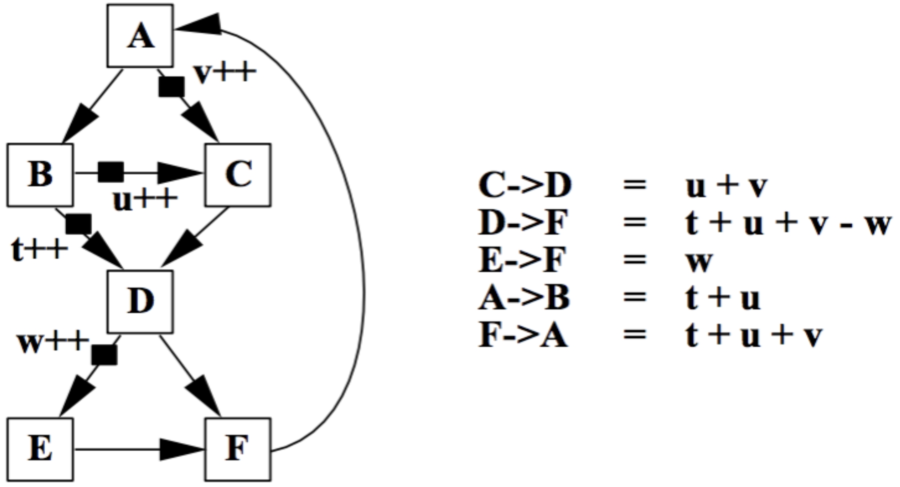
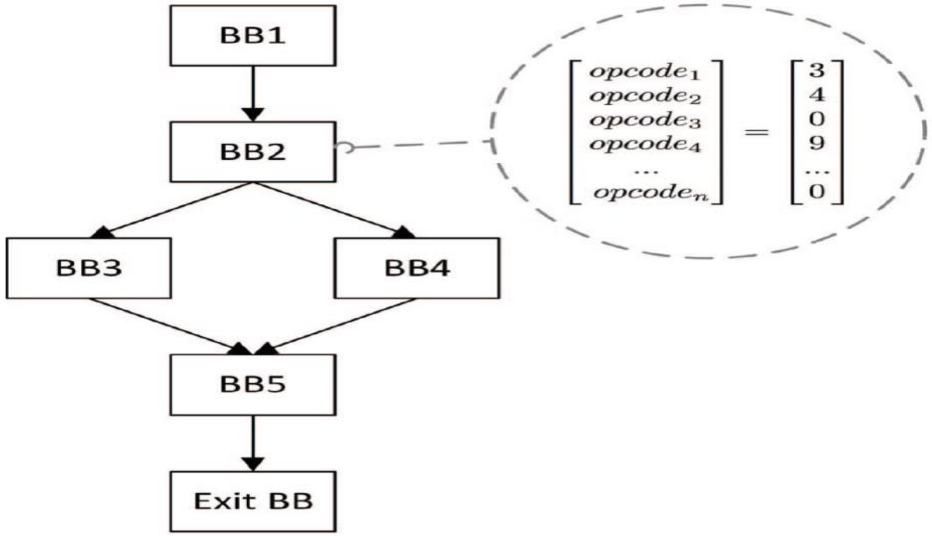
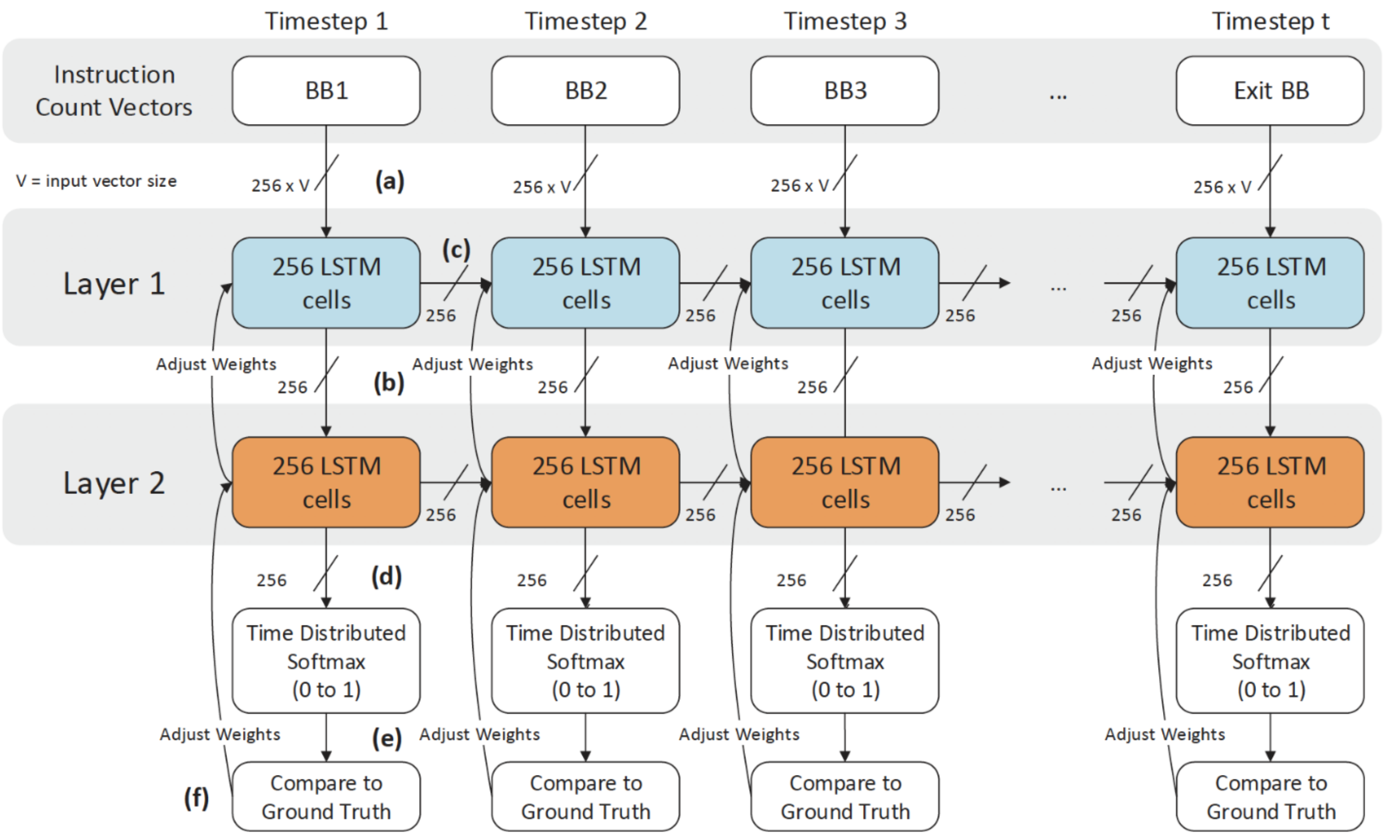


Figure 3. Instrumentation for edge profiling.



LSTM Architecture



Programs – SPEC CPU20

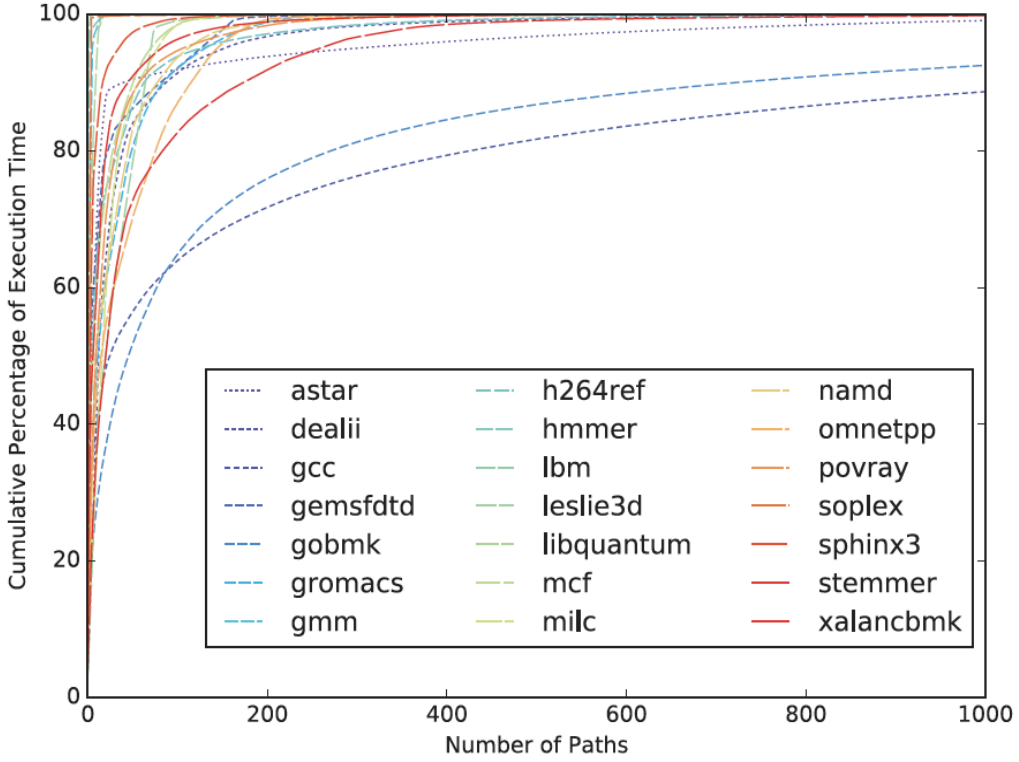


Fig. 9: Paths responsible for cumulative runtime

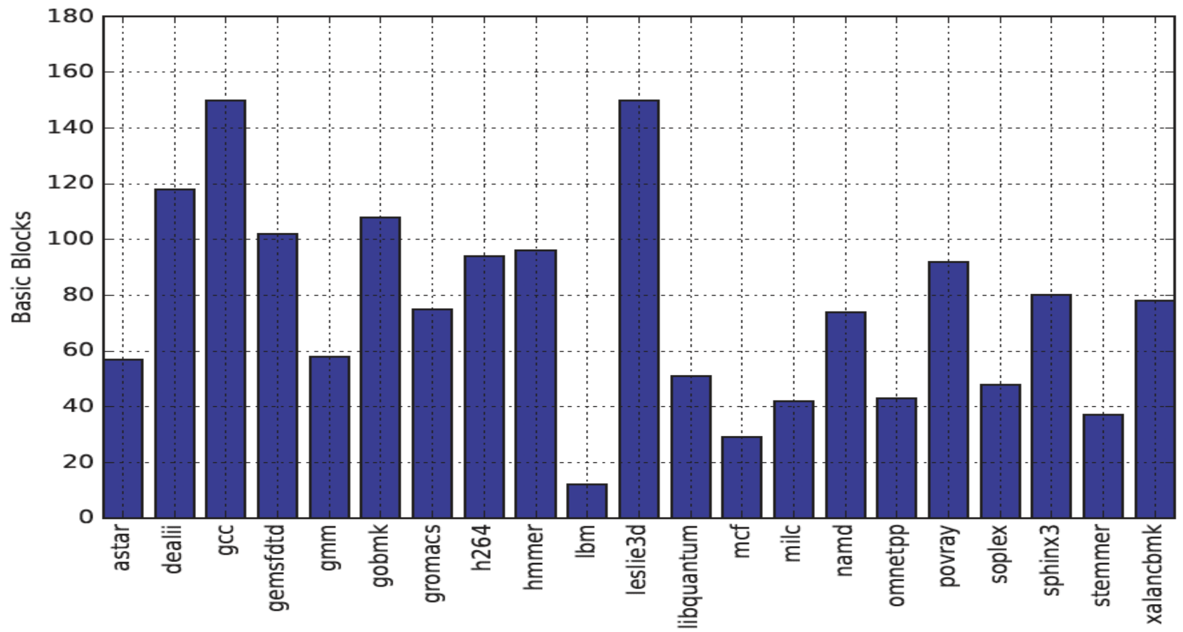


Fig. 11: Max path length by program

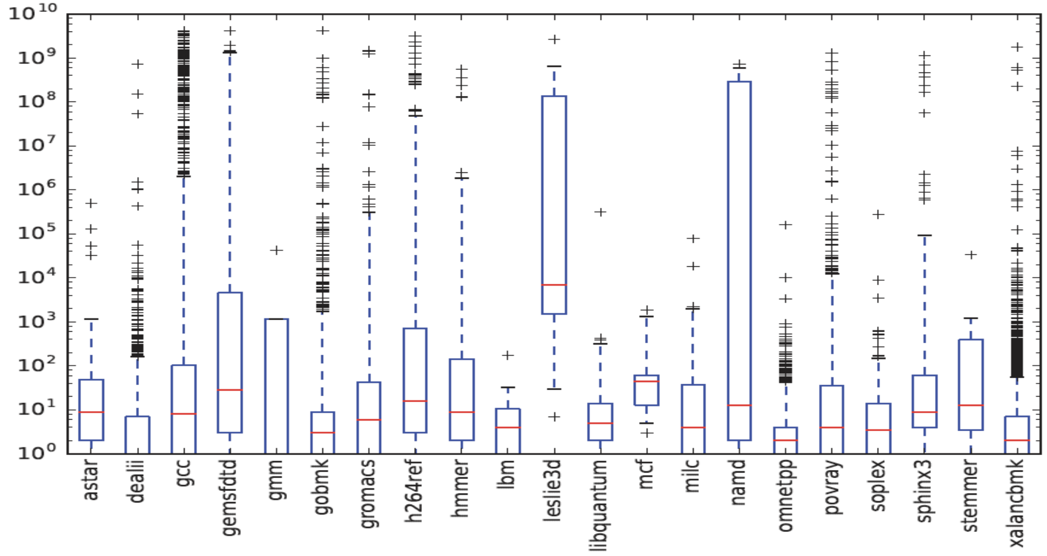


Fig. 10: Path counts per function

Logistic regression - B&W static path classifier

- Removed Features specific to java code
- Added IR specific feature
- Hand crafted features
- One feature vector per path
- B& W model – 0.83 AUROC,   Crystal Ball – 0.85

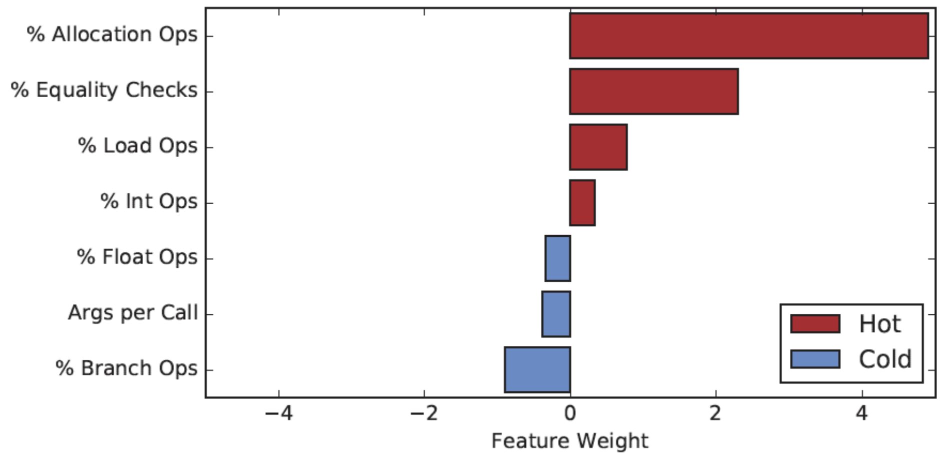


Fig. 15: Most important feature weights

Results -

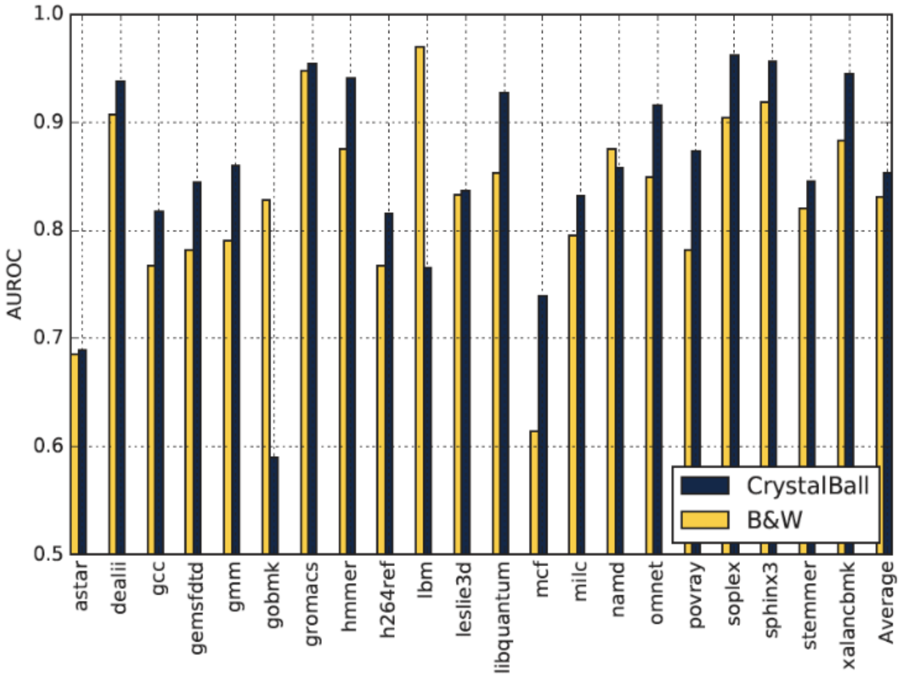


Fig. 12: AUROC by program

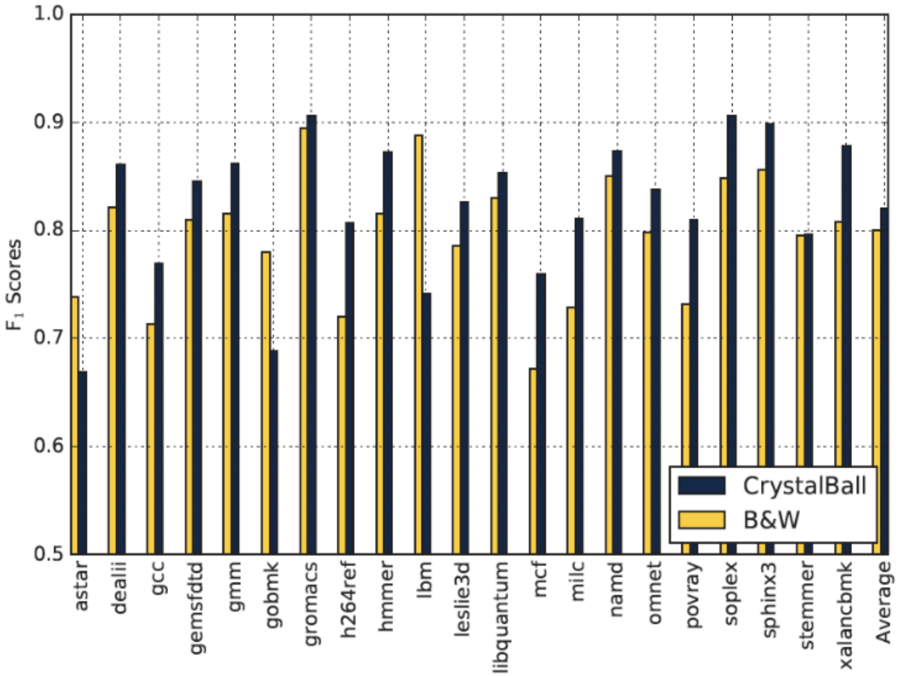


Fig. 13: F_1 scores by program

Future Work/Caveats

- Although AUROC is best among the shown measure, greater AUROC value doesn’t guarantee better model.
- Actual improvement in runtime behavior of a program?
- LSTM can just be used for feature extraction
- Novelty detection problem – SVM, K- Means
- Various Optimization flags and IR combination can be tried out.

Questions?